REMARKS

Claims 1-36 are pending in the application.

Claim rejections under 35 USC §103(a).

I. Independent Claims

Independent claims 1, 18, 31, and 34 stand rejected under 35 USC §103(a) as being unpatentable over Bunnel (U.S. Patent No. 5,594,903) in view of Galpin (U.S. Patent No. 7,043,728).

The Examiner stated:

> With respect to claim 1, the Bunnel et. al, reference teaches separating the program into computation segments (column 3, lines 9-22); compiling source code ... to generate two code sections (column 7, lines 8-27); executing each of the sections in a different computational domain ... (column 1, lines 28-50); (column 2, lines 30-46) and (column 16, line 65 - line 17, column 10).

However, Applicants note that column 3, lines 9-22 of Bunnel does not separate a program into "computation segments". Columm 3, lines 9 - 18, state:

> The operating system, upon execution by the processor, provides for the reservation of a first portion of the memory address space for support and application programs, ... a second portion for dynamic allocation and recovery by the operating system as necessary for the execution of support and application programs, and **a third portion,** located within said second portion, **for the static storage**, at predefined addresses, **of the executable code segments of the support and application programs** [emphasis supplied].

These "portions" in Bunnel are thus not separate "computation segments", since only the third portion of memory space in Bunnel is used for "computation". The "executable code segments" referred to above are not segments of a single program, as recited in Applicants' claim 1, but only single segments from each of several possible "support and application programs".

In column 7, at lines 10-16, Bunnel states:

> Program segments typically fall into one of two classes: code and data. Code segments, typically one per application program,

> contain the actual executable code of the application program. A data segment loaded simultaneous with the loading of a code segment permits image establishment of the predefined and preinitialized program data structures.

Thus, at lines 11-12, Bunnel states "**Code segments, typically one per application program**, contain the actual executable code of the application program". Thus, Bunnel makes it clear that a given program is not divided into multiple "computation segments", since "code" is required for computation, and there is clearly only one code section per application in Bunnel's system.

Thus Bunnel does not "compil[e] source code ... to generate two code sections", as claimed by Applicants, since Bunnel clearly states that only one of the two classes of "program segments" is executable, and only executable code can constitute a "computation segment".

Applicants can find nothing in either column 1, lines 28-50, or in column 2, lines 30-46, that mentions "executing each of the sections in a different computational domain". In column 16, line 65 - line 17, column 10, Bunnel states, in part:

> If, however, the expand flag was set, a control state 152 is entered instead. Here, the OS boot loader directly allocates the required size of the ".bss" segment without requiring the segment to be actually downloaded."

Applicants suggest that there is nothing in the above section directed to the execution of code sections "in a different computational domain".

Thus, Applicants maintain that Bunnel neither teaches nor suggests any of the claimed elements of "separating the program into computation segments"; "compiling source code ... to generate two code sections"; or "executing each of the sections in a different computational domain", as recited in claim 1. Therefore, it is irrelevant as to whether the Galpin reference teaches the additional claim elements suggested by the Examiner, since Galpin is used as a supplementary reference (rather than cumulatively) in conjunction with a reference (i.e., Bunnel) that does not supply a number of critical elements of Applicants' claim 1.

Even though Applicants believe that, for at least the above reasons, it is not necessary to distinguish claim 1 over the Galpin reference, Applicants, nevertheless, also point out below that this reference does not provide support for each of the elements suggested by the Examiner.

The Examiner states that the Galpin reference teaches "generating comparison code for comparing results produced by the execution of the two code sections (column 2, lines 29-48) and (column 3, lines 21-30)". However, Applicants note that the referenced section in Galpin makes no mention of "generating comparison code". Galpin only describes (directly) "comparing *states* of the first and second processes". Applicants assert that Galpin does not teach the use of generating comparison code to compare the *results* of the execution of two processes, but rather, as noted, instead only compares the *states* of the processes rather than the *results* of execution of the processes. In fact, Galpin specifically indicates that "one *aspect* [a *characteristic* of the invention, as opposed to an alternative embodiment] of the invention calls for comparing *flags* set during execution by the interrupt handlers of each process" (column 2, lines 37-39) [emphasis supplied].

The Examiner further states that the motivation to combine the Bunnel and Galpin references includes "effectuating loose coupling of the processes". However, even if, *arguendo*, the cited Bunnel and Galpin references contained teachings which provided each of the elements recited in Applicants' claim 1, the Examiner states a motivation which simply could not result in Applicants' invention as claimed. There is no "loose coupling" of Applicants' claimed redundant processes. The claimed processes are simply *not* coupled, and this is an additional distinction with respect to the cited references.

Furthermore, the Galpin reference states, at column 6, lines 7-12:

> As noted above, devices 10, 12 on both sides of the redundant pair have synchronization, or sync, lines 38 running between them that are used to keep their operation synchronized. This enforces loose coupling between the devices 10, 12 and, more particularly, between their respective processing sections 18, 20.

Thus Applicants maintain that the required "loose coupling" taught by Galpin negates the applicability of the reference to Applicants' claimed invention, in which there is *no* coupling of the redundant processes, either claimed, or indicated in the Specification.

Independent claim 18 also stands rejected under 35 USC §103(a) as being unpatentable over Bunnel and Galpin for essentially the same reasons as claims 1 and 2. Therefore, Applicants assert the same arguments with respect to claim 18 as those set forth above with respect to claim 1. Accordingly, Applicants believe that claim 18 patentably distinguishes over the cited references.

Independent claims 31 and 34 also stand rejected under 35 USC §103(a) as being unpatentable over Bunnel and Galpin for essentially the same reasons as claim 1. Therefore, Applicants assert the same arguments with respect to claims 31 and 34 as those set forth above with respect to claim 1. Accordingly, Applicants believe that claims 31 and 34 patentably distinguish over the cited references.

II. Dependent Claims

With respect to the motivation to combine the references, as required for each of the present rejections, Applicants note the following reasons (in addition to the reasons that each combination does not teach or suggest each of the elements in any of the claims to which the references are applied) why the Examiner's suggested motivation to combine, and the functionality of the applied reference is inapplicable to each combination:

Claims 2 and 35:

The Examiner states that "... Bunnel et. al. teaches wherein said computational domain comprises a time domain (column 9, lines 5-12)".

Bunnel, at column 9, lines 5-12, states:

> However, where the size of transient program area is constrained, or where the access response time must be short, it is undesirable to have to duplicate instances of each program into the transient program area in order to permit execution. Thus, the present invention provides the ability to maintain the code

> segments of programs not in a file system format, but rather in the stripped code segments area 78 in a format that may be directly executed.

The above paragraph provides no indication of a computational domain comprising a time domain. It simply addresses program formatting. Thus claims 2 and 35 cannot be rendered obvious by the unrelated teaching.

Claims 3, 24, and 32:

The Examiner states that "...Bunnel et. al. teaches wherein compiling the source code to schedule execution thereof so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections (column 3, lines 3-18), (column 3, lines 42-47) and (column 8, lines 31-36)".

Applicants point out that neither the term "clock cycle" nor even the word "clock" appears anywhere in the Bunnel reference. Thus claims 3, 24, and 32 cannot be rendered obvious by the unrelated teaching.

Claims 4 and 25:

The Examiner states that "The de Bonet reference teaches wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of duration of disruptive events causing said computational errors (column 7, lines 33-42)."

de Bonet states, at column 7, lines 33-42:

> Faults caused by overlong execution time of a protected software component 320 may also be detected by embodiments of the present invention. These types of faults are typically caused by code within the protected software component 320 which places the software system into an infinite loop, or causes portions of code to execute for an unusually long time. Certain embodiments of the present invention detect a fault of this type by comparing the time taken to execute protected software component 320 with a predetermined time period.

The above section in the de Bonet reference discusses program execution 'hanging' in either an infinite loop or executing "for an unusually long time". This paragraph bears no relation to predetermining a minimum number of processor clock cycles as a function of statistical properties of the duration of a disruptive event. Thus claims 4 and 25 cannot be rendered obvious by the unrelated teaching.

Claims 5, 6, and 36:

The Examiner states that "The Lajolo reference teaches wherein said computational domain comprises a spatial domain (column 3, lines 25-48)".

Column 3, lines 25-48, of the Lajolo reference state, in part:

> ...One of the key issues when using an electromagnetic simulator is the gridding process or, in other terms, how the structure under analysis is discretized and decomposed in spatial basis functions. The goal was to have a convergence of the simulation process on a 1 .mu.m width...

In view of the above, Applicants assert that there is nothing in column 3, lines 25-48 of the Lajolo reference that addresses the subject of computational domains. Lajolo is concerned with "how [a] structure under analysis is discretized". Thus claims 5, 6, and 36 cannot be rendered obvious by the unrelated teaching.

Claim 6:

The Examiner states that "...Bunnel et. al. teaches wherein the compiling step includes compiling the source code such that each of the code sections is executed using separate resources of the processor (column 3, lines 3-22) and (column 14, lines 20-34)".

In column 3, lines 3-22, Bunnel states, in part:

> ...a third portion, located within said second portion, for the static storage, at predefined addresses, of the executable code segments of the support and application programs.

Thus Bunnel compiles *all* source code into a *single* portion of memory.

In column 14, lines 20-34, Bunnel states, in part:

> ...If this operating system kernel is to be downloaded to a computer system 10, then the uninitialized portion of the data segment, specifically the ".bss" segment, is preferably deleted from the kernel image....

Here, Bunnel addresses loading an operating system kernel, and fails to mention anything related to the subject of executing code sections using separate resources of a processor. Thus, for at least the above reasons, claim 6 cannot be rendered obvious by the unrelated teachings.

Claim 7, 8, and 27:

The Examiner states that "the Kane et. al. reference teaches wherein said resources comprise functional units and partitioned registers", and adds that the Kane reference provides "the motivation for using a different set of functional units and partitioned registers of the processor is achieving a high degree of pipelining". Applicants assert that the Kane reference has no significant relationship (if any at all) to Applicant's claimed invention, or to claims 26 and 33, which utilize functional units and partitioned registers for the purpose of error detection, which purpose and related function is not related in any manner to "pipelining". If "pipelining" is in fact the motivation supplied by the Kane reference, then, Applicants assert that the system resulting from the cited combination of references cannot render claims 7, 8, and 27 obvious, given the unrelated teaching and non-relevant motivation to combine.

Claims 9, 10, 11, 13, 20, 21, and 29:

With respect to claims 9 and 28, the Examiner cites the Merkey reference and then states that "the motivation for wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections is to improve system performance".

Applicants note that the Merkey reference is concerned with partitioning *disk* storage devices, and fails to even mention partitioned registers. Thus claims 9 and 28 cannot be rendered obvious by the unrelated teaching.

With respect to claims 10 and 20, the Examiner cites the Merkey reference as teaching "wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed (column 12, lines 42-47)".

The cited section of Merkey discusses 'registering' device objects in Linux, and does not mention anything related to "executing comparison code". Thus claims 10 and 20 cannot be rendered obvious by the unrelated teaching.

With respect to claims 11 and 29, the Examiner states that "the Galpin references teaches wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are each executed by a different set of functional units ...".

Applicant notes that the cited sections in Galpin discuss "sender" and "listener" processors that "synchronously step through sequential schedules". This reference is thus entirely off-point with respect to Applicants' claims 11 and 29, since Applicants' claimed system operates entirely *asynchronously*. Thus claims 11 and 29 cannot be rendered obvious by the unrelated teaching.

Claims 15 and 16:

With respect to claim 15, the Examiner states that "the Oates reference teaches the step of optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections (column 8, line 62 - column 9, line 11)".

Column 8, line 62 - column 9, line 11, of the Oates reference state, in part:

> In other aspects, the invention provides an apparatus for efficiently computing a .GAMMA.-matrix as described above, e.g., in hardware. The system includes two registers, one associated with each of l.sup.th and k.sup.th users. The registers hold elements of the short code sequences associated with the respective user such that alignment of the short code sequence loaded in one register can be shifted relative to that of the other register by m elements. Associated with each of the foregoing registers is one additional register storing mask sequences....

Applicants maintain that this section of the Oates reference deals with the subject of "efficiently computing a .GAMMA.-matrix", and thus bears no relationship to Applicants' claim 15, which recites "optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections". Thus claim 15 cannot be rendered obvious by the unrelated teaching.

With respect to claim 16, the Examiner states that "the Bunnel reference teaches wherein the compiling step employs code reorganization to dynamically translate the source code into the two code sections" (column 6, lines 10-20) and (column 8, lines 28-40)".

In the above-cited sections, Bunnell respectively discusses "in general, the remaining RAM memory, generally referenced by the reference numerals 50, 50' is utilized as the transient program execution area" and "memory transfer requests directed to the logical interface corresponding to the pseudo-disk device driver results in a transfer of data fully within the main memory 14, though logically consistent with a disk drive paradigm". Neither of these sections is related to using "code reorganization to dynamically translate the source code into [the] two code sections", as recited in Applicants' claim 16. Thus claim 16 cannot be rendered obvious by the unrelated teaching.

Claim 17:

With respect to claim 17, the Examiner states that "the de Bonet reference teaches wherein the step of compiling the source code is performed by incrementally translating the source code (column 11, lines 47-67)".

Above, the examiner cites claim 13 in the de Bonet reference, which states:

> A data processing system readable medium, comprising code containing instructions translatable for: designating a first software component of a computer program as protected; wherein the first software component comprises a series of instructions to be executed on a central processing unit (CPU); saving a state of the computer program before executing the first software component; associating a remedial software component with the first software component; executing the first software component; determining whether a fault occurred during execution of the first

> software component, wherein the fault is caused by one or more programming defects within the first software component; and if the fault is detected, terminating the execution of the first software component; restoring the computer program to the previously saved state; and executing the remedial software component.

Applicants note that the "translatable" instructions referred to above are not related to the step of compiling the source code by incrementally translating the source code, as recited in Applicants' claim 17. Thus claim 17 cannot be rendered obvious by the unrelated teaching.

Claim 19:

Claim 19 was rejected for essentially the same reason as claim 2. Applicants thus submit the same argument as provided above with respect to claim 2, and assert, for the same reasons set forth therein, that claim 19 is patentable over the cited references.

Claim 22:

With respect to claim 22, the Examiner states "the Brown reference teaches an optimizer for modifying the output of the compiler (column 4, lines 7-27)".

Column 4, lines 7-27 of the Brown reference state, in part:

> ...In particular, output element 157 outputs a match to a predictor 158 for providing a confidence level and the output match to an optimizer 159 for optimizing a solution from the input match. The inference engine 155 is coupled to a knowledge base 156 for receiving created frames and managing frames and slots. The knowledge base accepts input from input/output interface 151 and outputs historical data to output 157...

Applicants maintain, in view of the above section, that Brown does not teach the use of "an optimizer for modifying the output of the compiler *to schedule execution of [the] redundant code*. Since Brown's purpose and function are entirely different than that claimed by Applicants in claim 22, Applicants thus submit that claim 22 cannot be rendered obvious by the unrelated teaching.

Claim 23:

The examiner states that the cited combination of references provides a "motivation for an optimizer for configuring is for an improvement in efficiency of a computer maintenance engineer". Applicants note that their claimed system (including claim 22) has absolutely nothing to do with improving the "efficiency of a computer maintenance engineer". Applicants assert that this statement so significantly mis-characterizes their claimed invention as to indicate that the cited combination of references must be inapplicable.

Claims 26 and 33:

With respect to claims 26 and 33, the Examiner cited the Kane reference as providing "the motivation for using a different set of functional units and partitioned registers of the processor is achieving a high degree of pipelining". Applicants assert that the Kane reference has no significant relationship (if any at all) to Applicant's claimed invention, or to claims 26 and 33, which utilize functional units and partitioned registers for the purpose of error detection, which purpose and related function is not related in any manner to "pipelining". If "pipelining" is in fact the motivation supplied by the Kane reference, then, Applicants assert that the system resulting from the cited combination of references cannot render claims 26 and 33 obvious, given the unrelated teaching and non-relevant motivation to combine.

III. Summary

Applicants note that, in order to combine references, there must be at least *some* relationship between the structure or function of the applied reference and the manner in which the claimed system actually operates. In each case noted above, Applicants assert that the stated motivation to combine the cited references cannot meet this minimum requirement, because each motivation to combine, as suggested by the Examiner, indicates that the applied reference operates in a significantly and patentably different manner than that in which Applicants' claimed invention functions, as described in their Specification.

This significant lack of relevance between Applicants' claimed system and the system which would result from applying the motivation suggested by the Examiner is further indicative of the lack of both relevance and applicability of the applied references, and further distinguishes Appellants' claimed invention over the references.

In each of the rejections set forth in the present Office Action, Applicants believe that the combination suggested by the Examiner is simply not similar to Applicants' claimed invention, either in structure, or in function, and therefore such a combination cannot be used to show obviousness of either of these claims or of the claims depending therefrom.

For at least the reasons enumerated above, Applicants believe that independent claims 1, 18, 31, and 34 are allowable over the cited art. Since dependent claims 2-17, 19-30, 32, 33, 35, and 36 incorporate the limitations of the respective independent claims, these claims should also be allowable.

Respectfully submitted,

LATHROP & GAGE L.C.

By: E. Michael Byorick

E. Michael Byorick Reg. No. 34,131
4845 Pearl East Circle, Suite 300
Boulder, Colorado 80301
Telephone: (720) 931-3000
Facsimile: (720) 931-3001